Part 1: Feature extraction

Q1

For unigram bag-of-words features, I first tranform all the words into lowercase, then discard stop words. I count all occurances of each word. Finally, I normalize the occurances by divding the number of words in the sentence.

Q2

For bigram bag-of-word features, what I do is similar to UnigramFeatureExtractor. After transforming words into lowercase, I discard adjacent pairs of words if both words are stop words. I also all occurances of each word and perform normalization.

Part 2: Implementing Logistic Regression Classifier

Q3

```
(python37) -sh-4.2$ python sentiment_classifier.py --model LR --feats UNIGRAM --no_run_on_t
est
Namespace(blind_test_path='data/test-blind.txt', dev_path='data/dev.txt', feats='UNIGRAM',
model='LR', run_on_test=False, test_output_path='test-blind.output.txt', train_path='data/t
rain.txt')
6920 / 872 / 1821 train/dev/test examples
[nltk_data] Downloading package stopwords to
[nltk_data]     /users/PAS0027/trainsn/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
epoch 0, loss: 0.585136, accuracy: 0.685838
epoch 1, loss: 0.424333, accuracy: 0.832225
epoch 2, loss: 0.351743, accuracy: 0.879913
epoch 3, loss: 0.306858, accuracy: 0.904191
epoch 4, loss: 0.273692, accuracy: 0.922254
epoch 5, loss: 0.249019, accuracy: 0.937139
=====Train Accuracy=====
Accuracy: 6598 / 6920 = 0.953468
Precision (fraction of predicted positives that are correct): 3389 / 3490 = 0.971060; Recal
l (fraction of true positives predicted correctly): 3389 / 3610 = 0.938781; F1 (harmonic me
an of precision and recall): 0.954648
=====Dev Accuracy=====
Accuracy: 678 / 872 = 0.777523
Precision (fraction of predicted positives that are correct): 338 / 426 = 0.793427; Recall
(fraction of true positives predicted correctly): 338 / 444 = 0.761261; F1 (harmonic mean o
f precision and recall): 0.777011
Time for training and evaluation: 6.00 seconds
```

Q4

```
(python37) -sh-4.2$ python sentiment_classifier.py --model LR --feats BIGRAM --no_run_on_te
st
Namespace(blind_test_path='data/test-blind.txt', dev_path='data/dev.txt', feats='BIGRAM', m
odel='LR', run_on_test=False, test_output_path='test-blind.output.txt', train_path='data/tr
ain.txt')
6920 / 872 / 1821 train/dev/test examples
[nltk_data] Downloading package stopwords to
[nltk_data]     /users/PAS0027/trainsn/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
epoch 0, loss: 0.647803, accuracy: 0.662428
epoch 1, loss: 0.458015, accuracy: 0.914740
epoch 2, loss: 0.365047, accuracy: 0.961272
epoch 3, loss: 0.304401, accuracy: 0.978757
epoch 4, loss: 0.261083, accuracy: 0.988873
epoch 5, loss: 0.228329, accuracy: 0.994075
=====Train Accuracy=====
Accuracy: 6906 / 6920 = 0.997977
Precision (fraction of predicted positives that are correct): 3605 / 3614 = 0.997510; Recal
l (fraction of true positives predicted correctly): 3605 / 3610 = 0.998615; F1 (harmonic me
an of precision and recall): 0.998062
=====Dev Accuracy=====
Accuracy: 638 / 872 = 0.731651
Precision (fraction of predicted positives that are correct): 353 / 496 = 0.711694; Recall
(fraction of true positives predicted correctly): 353 / 444 = 0.795045; F1 (harmonic mean o
f precision and recall): 0.751064
Time for training and evaluation: 11.45 seconds
```

Q5

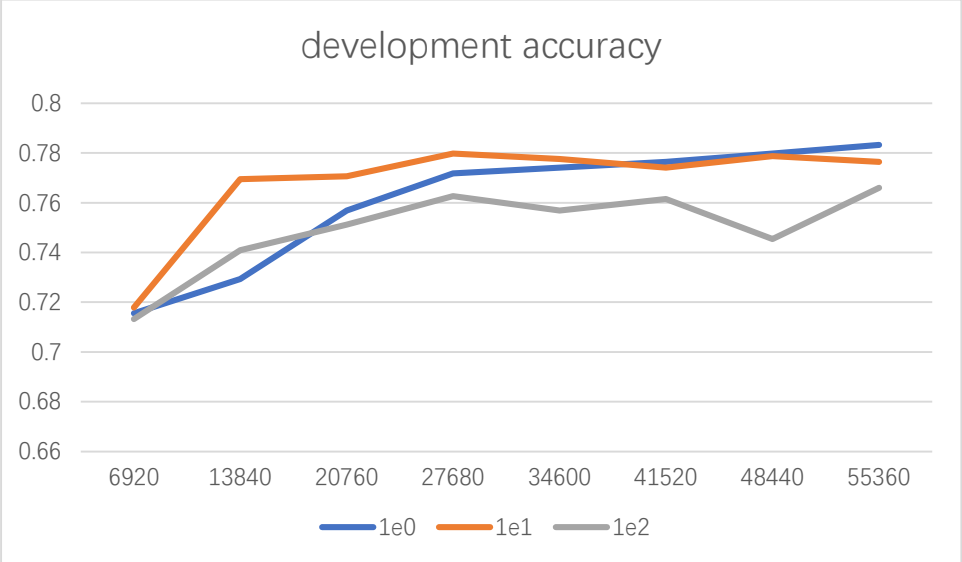
```
(python37) -sh-4.2$ python sentiment_classifier.py --model LR --feats BETTER --no_run_on_te
st
Namespace(blind_test_path='data/test-blind.txt', dev_path='data/dev.txt', feats='BETTER', m
odel='LR', run_on_test=False, test_output_path='test-blind.output.txt', train_path='data/tr
ain.txt')
6920 / 872 / 1821 train/dev/test examples
[nltk_data] Downloading package stopwords to
[nltk_data]     /users/PAS0027/trainsn/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
epoch 0, loss: 0.574881, accuracy: 0.701445
epoch 1, loss: 0.316685, accuracy: 0.912572
epoch 2, loss: 0.227377, accuracy: 0.961994
epoch 3, loss: 0.177481, accuracy: 0.983382
=====Train Accuracy=====
Accuracy: 6878 / 6920 = 0.993931
Precision (fraction of predicted positives that are correct): 3597 / 3626 = 0.992002; Recal
l (fraction of true positives predicted correctly): 3597 / 3610 = 0.996399; F1 (harmonic me
an of precision and recall): 0.994196
=====Dev Accuracy=====
Accuracy: 682 / 872 = 0.782110
Precision (fraction of predicted positives that are correct): 373 / 492 = 0.758130; Recall
(fraction of true positives predicted correctly): 373 / 444 = 0.840090; F1 (harmonic mean o
f precision and recall): 0.797009
Time for training and evaluation: 10.77 seconds
```

To obtain a better featrue, I combine unigrams with bigrams. Similar to UnigramFeatureExtractor, I keep the normalized words frequiences and discrad stop words.

Q6

The best feature setting I obtained is the better feature in Q5. Here are the training loss and development accuracy plot.

(When step size is 1e2, the training loss is nan.)



What we can see:

1. On the training set, starting from a small step size, when we increase it, we can see the training loss decreases faster. However, when the step size is too large, the model would explode, reflected by the nan training loss.
2. On the validation set, when we use a very large step size, it would be very hard for the model to achive a good local optimal.
3. On the validation set, using a small step size would make the model converge slower, but when we train the model for a long time, it gives us the best performance.